

Comprehensive Python Cheatsheet

=====

<sup>[Download text file](<https://raw.githubusercontent.com/gto76/python-cheatsheet/master/README.md>)

or [Fork me on GitHub](<https://github.com/gto76/python-cheatsheet>).

</sup>

![Monty Python](web/image_888.jpeg)

Main

```
```python
if __name__ == '__main__':
 main()
```
```

List

```
```python
<list> = <list>[from_inclusive : to_exclusive : step_size]
<list>.append(<el>)
<list>.extend(<collection>)
<list> += [<el>]
<list> += <collection>
```
```

```
```python
<list>.sort()
<list>.reverse()
<list> = sorted(<collection>)
<iter> = reversed(<list>)
```
```

```
```python
sum_of_elements = sum(<collection>)
elementwise_sum = [sum(pair) for pair in zip(list_a, list_b)]
sorted_by_second = sorted(<collection>, key=lambda el: el[1])
sorted_by_both = sorted(<collection>, key=lambda el: (el[1], el[0]))
flattened_list = list(itertools.chain.from_iterable(<list>))
list_of_chars = list(<str>)
product_of_elems = functools.reduce(lambda out, x: out * x, <collection>)
no_duplicates = list(dict.fromkeys(<list>))
```
```

```
```python
index = <list>.index(<el>) # Returns first index of item.
<list>.insert(index, <el>) # Inserts item at index and moves the rest to the right.
<el> = <list>.pop([index]) # Removes and returns item at index or from the end.
<list>.remove(<el>) # Removes first occurrence of item.
<list>.clear() # Removes all items.
```
```

VISIT...

LANZAROTE
Caliente.COM

Dictionary

```
```python
<view> = <dict>.keys()
<view> = <dict>.values()
<view> = <dict>.items()
```

```python
value = <dict>.get(key, default) # Returns default if key does not exist.
value = <dict>.setdefault(key, default) # Same, but also adds default to dict.
<dict> = collections.defaultdict(<type>) # Creates a dictionary with default value of
type. #
<dict> = collections.defaultdict(lambda: 1) # Creates a dictionary with default value 1.
```

```python
<dict>.update(<dict>) # Or: dict_a = {**dict_a, **dict_b}.
<dict> = dict(<list>) # Initiates a dict from list of key-value
pairs.
<dict> = dict(zip(keys, values)) # Initiates a dict from two lists.
<dict> = dict.fromkeys(keys [, value]) # Initiates a dict from list of keys.
```

```python
value = <dict>.pop(key) # Removes item from dictionary.
{k: v for k, v in <dict>.items() if k in keys} # Filters dictionary by keys.
```
```

Counter

```
```python
>>> from collections import Counter
>>> colors = ['blue', 'red', 'blue', 'yellow', 'blue', 'red']
>>> counter = Counter(colors)
Counter({'blue': 3, 'red': 2, 'yellow': 1})
>>> counter.most_common()[0][0]
'blue'
```
```

Set

```
```python
<set> = set()
<set>.add(<el>)
<set>.update(<collection>)
<set> |= {<el>}
<set> |= <set>
```

```python
<set> = <set>.union(<coll.>) # Or: <set> | <set>
<set> = <set>.intersection(<coll.>) # Or: <set> & <set>
<set> = <set>.difference(<coll.>) # Or: <set> - <set>
```

```
<set> = <set>.symmetric_difference(<coll.>) # Or: <set> ^ <set>
<bool> = <set>.issubset(<coll.>) # Or: <set> <= <set>
<bool> = <set>.issuperset(<coll.>) # Or: <set> >= <set>
...

```

```
```python
<set>.remove(<el>) # Throws error.
<set>.discard(<el>) # Doesn't throw error.
...

```

```
### Frozenset
#### Is hashable and can be used as a key in dictionary.
```python
<frozenset> = frozenset(<collection>)
...

```

## Range

```

```python
range(to_exclusive)
range(from_inclusive, to_exclusive)
range(from_inclusive, to_exclusive, step_size)
range(from_inclusive, to_exclusive, -step_size)
...

```

```
```python
from_inclusive = <range>.start
to_exclusive = <range>.stop
...

```

## Enumerate

```

```python
for i, el in enumerate(<collection> [, i_start]):
    ...
...

```

Named Tuple

```
-----
```python
>>> Point = collections.namedtuple('Point', 'x y')
>>> p = Point(1, y=2)
Point(x=1, y=2)
>>> p[0]
1
>>> p.x
1
>>> getattr(p, 'y')
2
>>> p._fields # Or: Point._fields
('x', 'y')
...

```

## Iterator

-----

```
```python
<iter> = iter(<collection>)
<iter> = iter(<function>, to_exclusive)
```
```

#### Reads input until it reaches an empty line:

```
```python
for line in iter(input, ''):
    ...
```
```

#### Same, but prints a message every time:

```
```python
from functools import partial
for line in iter(partial(input, 'Please enter value: '), ''):
    ...
```
```

### Next

**\*\*Returns next item. If there are no more items it raises exception or returns default if specified.\*\***

```
```python
<el> = next(<iter> [, default])
```
```

#### Skips first item:

```
```python
next(<iter>)
for element in <iter>:
    ...
```
```

## Generator

-----

**\*\*Convenient way to implement the iterator protocol.\*\***

```
```python
def step(start, step_size):
    while True:
        yield start
        start += step_size
```
```

```
```python
>>> stepper = step(10, 2)
>>> next(stepper), next(stepper), next(stepper)
(10, 12, 14)
```
```

## Type

-----

```
```python
<type> = type(<el>) # <class 'int'> / <class 'str'> / ...
```

```python
from numbers import Number, Integral, Real, Rational, Complex
<bool> = isinstance(<el>, Number)
```

```python
<bool> = callable(<el>)
```
```

## String

-----

```
```python
<str> = <str>.strip() # Strips all whitespace characters from both ends.
<str> = <str>.strip('<chars>') # Strips all passed characters from both ends.
```

```python
<list> = <str>.split() # Splits on any whitespace character.
<list> = <str>.split(sep=None, maxsplit=-1) # Splits on 'sep' str at most 'maxsplit'
times.
<str> = <str>.join(<list>) # Joins elements using string as separator.
```

```python
<str> = <str>.replace(old_str, new_str)
<bool> = <str>.startswith(<sub_str>) # Pass tuple of strings for multiple options.
<bool> = <str>.endswith(<sub_str>) # Pass tuple of strings for multiple options.
<int> = <str>.index(<sub_str>) # Returns first index of a substring.
<bool> = <str>.isnumeric() # True if str contains only numeric characters.
<list> = textwrap.wrap(<str>, width) # Nicely breaks string into lines.
```

Char
```python
<str> = chr(<int>) # Converts int to unicode char.
<int> = ord(<str>) # Converts unicode char to int.
```

```python
>>> ord('0'), ord('9')
(48, 57)
>>> ord('A'), ord('Z')
(65, 90)
>>> ord('a'), ord('z')
(97, 122)
```
```

## Regex

-----

```
```python
import re
<str> = re.sub(<regex>, new, text, count=0) # Substitutes all occurrences.
<list> = re.findall(<regex>, text) # Returns all occurrences.
<list> = re.split(<regex>, text, maxsplit=0) # Use brackets in regex to keep the matches.
<Match> = re.search(<regex>, text) # Searches for first occurrence of pattern.
<Match> = re.match(<regex>, text) # Searches only at the beginning of the
text.
<iter> = re.finditer(<regex>, text) # Returns all occurrences as match objects.
```

* **Parameter `flags=re.IGNORECASE` can be used with all functions.**
* **Parameter `flags=re.DOTALL` makes dot also accept newline.**
* **Use `r'\1` or `\\1` for backreference.**
* **Use `?` to make operators non-greedy.**
```

## Match Object

```
```python
<str> = <Match>.group() # Whole match.
<str> = <Match>.group(1) # Part in first bracket.
<tuple> = <Match>.groups() # All bracketed parts.
<int> = <Match>.start() # Start index of a match.
<int> = <Match>.end() # Exclusive end index of a match.
```
```

## Special Sequences

```
Use capital letter for negation.
```python
'\d' == '[0-9]' # Digit
'\s' == '[\t\n\r\f\v]' # Whitespace
'\w' == '[a-zA-Z0-9_]' # Alphanumeric
```
```

## Format

-----

```
```python
<str> = f'{<el_1>}, {<el_2>}'
<str> = '{}, {}'.format(<el_1>, <el_2>)
```

```python
>>> Person = namedtuple('Person', 'name height')
>>> person = Person('Jean-Luc', 187)
>>> f'{person.height:10}'
'      187'
>>> '{p.height:10}'.format(p=person)
'      187'
```
```

## General Options

```
```python
{<el>:<10>} # '<el>'
```

```
{<el>:>10}      # '      <el>'
{<el>:^10}       # '      <el>'
{<el>:->10}      # '-----<el>'
{<el>:>0}        # '<el>'
```

```

### ### String Options

\*\*`!r` calls object's repr() method, instead of format(), to get a string.\*\*

```
```python
{'abcde'!r:<10} # "'abcde'  "
```
```

```
```python
{'abcde':.3}    # 'abc'
{'abcde':10.3}  # 'abc      '
```

Number Options

```
```python
{1.23456:.3f} # '1.235'
{1.23456:10.3f} # ' 1.235'
```
```

```
```python
{ 123456:10,} # ' 123,456'
{ 123456:10_} # ' 123_456'
{ 123456:+10} # ' +123456'
{-123456:=10} # '- 123456'
{ 123456: } # ' 123456'
{-123456: } # '-123456'
```
```

```
```python
{65:c} # 'A'
{3:08b} # '00000011' -> Binary with leading zeros.
{3:0<8b} # '11000000' -> Binary with trailing zeros.
```
```

Float presentation types:

```
* **`f` - Fixed point: `.<precision>f`**
* **`%` - Percent: `.<precision>%`**
* **`e` - Exponent**
```

Integer presentation types:

```
* **`c` - character**
* **`b` - binary**
* **`x` - hex**
* **`X` - HEX**
```

Numbers

Basic Functions

```
```python
<num> = pow(<num>, <num>) # Or: <num> ** <num>
```



```
<real> = abs(<num>)
<real> = round(<real> [, ndigits])
'''

Constants
```python
from math import e, pi
'''

### Trigonometry
```python
from math import cos, acos, sin, asin, tan, atan, degrees, radians
'''

Logarithm
```python
from math import log, log10, log2
<float> = log(<real> [, base]) # Base e, if not specified.
'''

### Infinity, nan
```python
from math import inf, nan, isinf, isnan
'''

Or:
```python
float('inf'), float('nan')
'''

### Random
```python
from random import random, randint, choice, shuffle
<float> = random()
<int> = randint(from_inclusive, to_inclusive)
<el> = choice(<list>)
shuffle(<list>)
'''

Datetime

```python
from datetime import datetime
now = datetime.now()
now.month           # 3
now.strftime('%Y%m%d') # '20180315'
now.strftime('%Y%m%d%H%M%S') # '20180315002834'
<datetime> = datetime.strptime('2015-05-12 00:39', '%Y-%m-%d %H:%M')
'''

Arguments
-----
**`*` is the splat operator, that takes a list as input, and expands it into actual
```

positional arguments in the function call.**

```
```python
args = (1, 2)
kwargs = {'x': 3, 'y': 4, 'z': 5}
func(*args, **kwargs)
```
```

Is the same as:

```
```python
func(1, 2, x=3, y=4, z=5)
```
```

Splat operator can also be used in function declarations:

```
```python
def add(*a):
 return sum(a)
```
```

```
```python
>>> add(1, 2, 3)
6
```
```

And in few other places:

```
```python
>>> a = (1, 2, 3)
>>> [*a]
[1, 2, 3]
```
```

```
```python
>>> head, *body, tail = [1, 2, 3, 4]
>>> body
[2, 3]
```
```

Inline

```
-----
### Lambda
```python
lambda: <return_value>
lambda <argument_1>, <argument_2>: <return_value>
```
```

Comprehension

```
```python
<list> = [i+1 for i in range(10)] # [1, 2, ..., 10]
<set> = {i for i in range(10) if i > 5} # {6, 7, 8, 9}
<dict> = {i: i*2 for i in range(10)} # {0: 0, 1: 2, ..., 9: 18}
<iter> = (i+5 for i in range(10)) # (5, 6, ..., 14)
```
```

```
```python
```

```
out = [i+j for i in range(10) for j in range(10)]
'''

Is the same as:
```python
out = []
for i in range(10):
    for j in range(10):
        out.append(i+j)
'''

### Map, Filter, Reduce
```python
from functools import reduce
<iter> = map(lambda x: x + 1, range(10)) # (1, 2, ..., 10)
<iter> = filter(lambda x: x > 5, range(10)) # (6, 7, 8, 9)
<int> = reduce(lambda out, x: out + x, range(10)) # 45
'''

Any, All
```python
<bool> = any(<collection>)                       # False if empty.
<bool> = all(el[1] for el in <collection>)        # True if empty.
'''

### If - Else
```python
<expression_if_true> if <condition> else <expression_if_false>
'''

```python
>>> [a if a else 'zero' for a in (0, 1, 0, 3)]
['zero', 1, 'zero', 3]
'''

### Namedtuple, Enum, Class
```python
from collections import namedtuple
Point = namedtuple('Point', 'x y')
point = Point(0, 0)
'''

```python
from enum import Enum
Direction = Enum('Direction', 'n e s w')
Cutlery   = Enum('Cutlery', {'fork': 1, 'knife': 2, 'spoon': 3})
'''

```python
Warning: Objects will share the objects that are initialized in the dictionary!
Creature = type('Creature', (), {'p': Point(0, 0), 'd': Direction.n})
creature = Creature()
'''
```

## Closure

-----

**\*\*We have a closure in Python when:\*\***  
**\* \*\*A nested function references a value of its enclosing function and then\*\***  
**\* \*\*the enclosing function returns the nested function.\*\***

```
```python
def get_multiplier(a):
    def out(b):
        return a * b
    return out
```
```

```
```python
>>> multiply_by_3 = get_multiplier(3)
>>> multiply_by_3(10)
30
```
```

**\* \*\*If multiple nested functions within enclosing function reference the same value, that value gets shared.\*\***  
**\* \*\*To dynamically access functions first free variable use**  
**`'<function>.__closure__[0].cell_contents'`.**

```
Or:
```python
from functools import partial
<function> = partial(<function>, <argument_1> [, <argument_2>, ...])
```
```

```
```python
>>> multiply_by_3 = partial(operator.mul, 3)
>>> multiply_by_3(10)
30
```
```

**### Nonlocal**  
**\*\*If variable is assigned to anywhere in the scope, it is regarded as a local variable, unless it is declared as global or nonlocal.\*\***

```
```python
def get_counter():
    a = 0
    def out():
        nonlocal a
        a += 1
        return a
    return out
```
```

```
```python
>>> counter = get_counter()
>>> counter(), counter(), counter()
(1, 2, 3)
```
```

## Decorator

-----

**\*\*A decorator takes a function, adds some functionality and returns it.\*\***

```
```python
```

```
@decorator_name
```

```
def function_that_gets_passed_to_decorator():
```

```
...    ...
```

```
### Debugger Example
```

****Decorator that prints function's name every time it gets called.****

```
```python
```

```
from functools import wraps
```

```
def debug(func):
```

```
 @wraps(func)
```

```
 def out(*args, **kwargs):
```

```
 print(func.__name__)
```

```
 return func(*args, **kwargs)
```

```
 return out
```

```
@debug
```

```
def add(x, y):
```

```
 return x + y
```

```
...
```

**\* \*\*Wraps is a helper decorator that copies metadata of function add() to function out().\*\***

**\* \*\*Without it 'add.\_\_name\_\_' would return 'out'.\*\***

```
LRU Cache
```

**\*\*Decorator that caches function's return values. All function's arguments must be hashable.\*\***

```
```python
```

```
from functools import lru_cache
```

```
@lru_cache(maxsize=None)
```

```
def fib(n):
```

```
    return n if n < 2 else fib(n-1) + fib(n-2)
```

```
...
```

```
```python
```

```
>>> [fib(n) for n in range(10)]
```

```
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

```
>>> fib.cache_info()
```

```
CacheInfo(hits=16, misses=10, maxsize=None, currsize=10)
```

```
...
```

```
Parametrized Decorator
```

```
```python
```

```
from functools import wraps
```

```
def debug(print_result=False):
    def decorator(func):
        @wraps(func)
        def out(*args, **kwargs):
            result = func(*args, **kwargs)
            print(func.__name__, result if print_result else '')
            return result
        return out
    return decorator

@debug(print_result=True)
def add(x, y):
    return x + y
...
```

Class

```
-----
```python
class <name>:
 def __init__(self, a):
 self.a = a
 def __repr__(self):
 class_name = type(self).__name__
 return f'{class_name}({self.a!r})'
 def __str__(self):
 return str(self.a)

 @classmethod
 def get_class_name(cls):
 return cls.__name__
...
```

```
Constructor Overloading
```python
class <name>:
    def __init__(self, a=None):
        self.a = a
...
```

```
### Inheritance
```python
class Person:
 def __init__(self, name, age):
 self.name = name
 self.age = age

class Employee(Person):
 def __init__(self, name, age, staff_num):
 super().__init__(name, age)
 self.staff_num = staff_num
...
```

```
Comparable
* **If eq() method is not overridden, it returns `id(self) == id(other)`, which is the
```

same as `'self is other'`.`

`* **That means all objects compare not equal by default.**`

````python`

`class MyComparable:`

`def __init__(self, a):`

`self.a = a`

`def __eq__(self, other):`

`if isinstance(other, type(self)):`

`return self.a == other.a`

`return False`

`...`

`### Hashable`

`* **Hashable object needs both hash() and eq() methods and it's hash value should never change.**`

`* **Hashable objects that compare equal must have the same hash value, meaning default hash() that returns 'id(self)' will not do.**`

`* **That is why Python automatically makes classes unhashable if you only implement eq().**`

````python`

`class MyHashable:`

`def __init__(self, a):`

`self.__a = copy.deepcopy(a)`

`@property`

`def a(self):`

`return self.__a`

`def __eq__(self, other):`

`if isinstance(other, type(self)):`

`return self.a == other.a`

`return False`

`def __hash__(self):`

`return hash(self.a)`

`...`

`### Sequence`

`* **Methods do not depend on each other, so they can be skipped if not needed.**`

`* **Any object with defined getitem() is considered iterable, even if it lacks iter().**`

````python`

`class MySequence:`

`def __init__(self, a):`

`self.a = a`

`def __len__(self):`

`return len(self.a)`

`def __getitem__(self, i):`

`return self.a[i]`

`def __iter__(self):`

`for el in self.a:`

`yield el`

`...`

`### Callable`

````python`

`class Counter:`

`def __init__(self):`

```
 self.a = 0
 def __call__(self):
 self.a += 1
 return self.a
...

Copy
```python
from copy import copy, deepcopy
<object> = copy(<object>)
<object> = deepcopy(<object>)
...

Enum
----
```python
from enum import Enum, auto

class <enum_name>(Enum):
 <member_name_1> = <value_1>
 <member_name_2> = <value_2_a>, <value_2_b>
 <member_name_3> = auto()

 @classmethod
 def get_member_names(cls):
 return [a.name for a in cls.__members__.values()]
...

```python
<member> = <enum>.<member_name>
<member> = <enum>['<member_name>']
<member> = <enum>(<value>)
name      = <member>.name
value     = <member>.value
...

```python
list_of_members = list(<enum>)
member_names = [a.name for a in <enum>]
member_values = [a.value for a in <enum>]
random_member = random.choice(list(<enum>))
...

Inline
```python
Cutlery = Enum('Cutlery', ['fork', 'knife', 'spoon'])
Cutlery = Enum('Cutlery', 'fork knife spoon')
Cutlery = Enum('Cutlery', {'fork': 1, 'knife': 2, 'spoon': 3})
...

#### Functions can not be values, so they must be wrapped:
```python
from functools import partial
LogicOp = Enum('LogicOp', {'AND': partial(lambda l, r: l and r),
```



```
... 'OR' : partial(lambda l, r: l or r)})
```

## Exceptions

-----

```
```python
while True:
    try:
        x = int(input('Please enter a number: '))
    except ValueError:
        print('Oops! That was no valid number. Try again...')
    else:
        print('Thank you.')
        break
...

```

Raising exception:

```
```python
raise ValueError('A very specific message!')
...

```

### Finally

```
```python
>>> try:
...     raise KeyboardInterrupt
... finally:
...     print('Goodbye, world!')
Goodbye, world!
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
KeyboardInterrupt
...

```

System

Command Line Arguments

```
```python
import sys
script_name = sys.argv[0]
arguments = sys.argv[1:]
...

```

### Print Function

```
```python
print(<el_1>, ..., sep=' ', end='\n', file=sys.stdout, flush=False)
...

```

* **Use `file=sys.stderr` for errors.**

Pretty print:

```
```python
>>> from pprint import pprint
>>> pprint(dir())

```

```
['__annotations__',
 '__builtins__',
 '__doc__', ...]
...

Input Function
* **Reads a line from user input or pipe if present.**
* **The trailing newline gets stripped.**
* **The prompt string is printed to standard output before reading input.**

```python
<str> = input(prompt=None)
...

#### Prints lines until EOF:
```python
while True:
 try:
 print(input())
 except EOFError:
 break
...

Open Function
* **Opens file and returns a corresponding file object.**

```python
<file> = open(<path>, mode='r', encoding=None)
...

#### Modes:
* **`'r'` - Read (default).**
* **`'w'` - Write (truncate).**
* **`'x'` - Write or fail if the file already exists.**
* **`'a'` - Append.**
* **`'w+'` - Read and write (truncate).**
* **`'r+'` - Read and write from the beginning.**
* **`'a+'` - Read and write from the end.**
* **`'b'` - Binary mode.**
* **`'t'` - Text mode (default).**

#### Read Text from File:
```python
def read_file(filename):
 with open(filename, encoding='utf-8') as file:
 return file.readlines()
...

Write Text to File:
```python
def write_to_file(filename, text):
    with open(filename, 'w', encoding='utf-8') as file:
        file.write(text)
...

```

```
### Command Execution
```python
import os
<str> = os.popen(<command>).read()
```
```

```
#### Or:
```python
>>> import subprocess
>>> a = subprocess.run(['ls', '-a'], stdout=subprocess.PIPE)
>>> a.stdout
b'.\n..\nfile1.txt\nfile2.txt\n'
>>> a.returncode
0
```
```

```
### Recursion Limit
```python
>>> import sys
>>> sys.getrecursionlimit()
1000
>>> sys.setrecursionlimit(5000)
```
```

```
### Path
```python
from os import path, listdir
<bool> = path.exists(<path>)
<bool> = path.isfile(<path>)
<bool> = path.isdir(<path>)
<list> = listdir(<path>)
```
```

```
```python
>>> from glob import glob
>>> glob('../*.gif')
['1.gif', 'card.gif']
```
```

Pathlib

****This module offers classes representing filesystem paths with semantics appropriate for different operating systems.****

```
```python
from pathlib import Path
pwd = Path()
<Path> = Path('<path>' [, '<path>', <Path>, ...])
<Path> = <Path> / '<dir>' / '<file>'
```
```

```
```python
<iter> = <Path>.iterdir() # Returns all files in a dir.
<iter> = <Path>.glob('<pattern>') # Returns all matches.
```

```
<Path> = <Path>.resolve() # Makes path absolute.
<bool> = <Path>.exists()
<bool> = <Path>.is_dir()
<file> = <Path>.open()
'''

```python
<str>    = str(<Path>)              # Returns path as string.
<str>    = <Path>.name              # Final component.
<str>    = <Path>.stem              # Final component without extension.
<str>    = <Path>.suffix            # Final component's extension.
<Path>   = <Path>.parent            # Path without final component.
<tuple>  = <Path>.parts            # All components as strings.
'''
```

JSON

```
```python
import json
<str> = json.dumps(<object>, ensure_ascii=True, indent=None)
<object> = json.loads(<str>)
'''
```

#### To preserve order:

```
```python
from collections import OrderedDict
<object> = json.loads(<str>, object_pairs_hook=OrderedDict)
'''
```

Read File

```
```python
def read_json_file(filename):
 with open(filename, encoding='utf-8') as file:
 return json.load(file)
'''
```

### Write to File

```
```python
def write_to_json_file(filename, an_object):
    with open(filename, 'w', encoding='utf-8') as file:
        json.dump(an_object, file, ensure_ascii=False, indent=2)
'''
```

Pickle

```
```python
import pickle
<bytes> = pickle.dumps(<object>)
<object> = pickle.loads(<bytes>)
'''
```

### Read Object from File

```
```python
```

```
def read_pickle_file(filename):
    with open(filename, 'rb') as file:
        return pickle.load(file)
...
```

Write Object to File

```
```python
def write_to_pickle_file(filename, an_object):
 with open(filename, 'wb') as file:
 pickle.dump(an_object, file)
...
```

SQLite

-----

```
```python
import sqlite3
db = sqlite3.connect(<filename>)
...
db.close()
...
```

Read

```
```python
cursor = db.execute(<query>)
if cursor:
 <tuple> = cursor.fetchone() # First row.
 <list> = cursor.fetchall() # Remaining rows.
...
```

### Write

```
```python
db.execute(<query>)
db.commit()
...
```

Bytes

****Bytes object is immutable sequence of single bytes. Mutable version is called bytearray.****

```
```python
<bytes> = b'<str>'
<int> = <bytes>[<index>]
<bytes> = <bytes>[<slice>]
<bytes> = b''.join(<coll_of_bytes>)
...
```

### Encode

```
```python
<bytes> = <str>.encode(encoding='utf-8')
<bytes> = <int>.to_bytes(length, byteorder='big|little', signed=False)
<bytes> = bytes.fromhex(<hex>)
...
```

```
### Decode
```python
<str> = <bytes>.decode('utf-8')
<int> = int.from_bytes(<bytes>, byteorder='big|little', signed=False)
<hex> = <bytes>.hex()
```
```

```
### Read Bytes from File
```python
def read_bytes(filename):
 with open(filename, 'rb') as file:
 return file.read()
```
```

```
### Write Bytes to File
```python
def write_bytes(filename, bytes_obj):
 with open(filename, 'wb') as file:
 file.write(bytes_obj)
```
```

Struct

```
* **Module that performs conversions between Python values and a C struct, represented as a
Python bytes object.**
* **Machine's native type sizes and byte order are used by default.**
```

```
```python
from struct import pack, unpack, calcsize
<bytes> = pack('<format>', <value_1> [, <value_2>, ...])
<tuple> = unpack('<format>', <bytes>)
```
```

```
### Example
```python
>>> pack('>hhl', 1, 2, 3)
b'\x00\x01\x00\x02\x00\x00\x00\x03'
>>> unpack('>hhl', b'\x00\x01\x00\x02\x00\x00\x00\x03')
(1, 2, 3)
>>> calcsize('>hhl')
8
```
```

Format

```
#### For standard sizes start format string with:
```

```
* **`'='` - native byte order**
* **`'<` - little-endian**
* **`'>` - big-endian**
```

```
#### Use capital letter for unsigned type. Standard size in brackets:
```

```
* **`'x'` - pad byte**
* **`'c'` - char (1)**
* **`'h'` - short (2)**
```

```
* **`'i'` - int (4)**
* **`'l'` - long (4)**
* **`'q'` - long long (8)**
* **`'f'` - float (4)**
* **`'d'` - double (8)**
```

Array

****List that can only hold elements of predefined type. Available types are listed above.****

```
```python
from array import array
<array> = array(<typecode> [, <collection>])
```
```

Memory View

****Used for accessing the internal data of an object that supports the buffer protocol.****

```
```python
<memoryview> = memoryview(<bytes> / <bytearray> / <array>)
<memoryview>.release()
```
```

Deque

****A thread-safe list with efficient appends and pops from either side. Pronounced “deck”.****

```
```python
from collections import deque
<deque> = deque(<collection>, maxlen=None)
```
```

```
```python
<deque>.appendleft(<el>)
<deque>.extendleft(<collection>) # Collection gets reversed.
<el> = <deque>.popleft()
<deque>.rotate(n=1) # Rotates elements to the right.
```
```

Threading

```
```python
from threading import Thread, RLock
```
```

```
### Thread
```python
thread = Thread(target=<function>, args=(<first_arg>,))
thread.start()
...
```
```

```
thread.join()
...
```

```
### Lock
```python
lock = RLock()
lock.acquire()
...
lock.release()
...
```

## Hashlib

```

```python
>>> import hashlib
>>> hashlib.md5(<str>.encode()).hexdigest()
'33d0eba106da4d3ebca17fcd3f4c3d77'
...
```

Itertools

```
-----
* **Every function returns an iterator and can accept any collection and/or iterator.**
* **If you want to print the iterator, you need to pass it to the list() function!**
```

```
```python
from itertools import *
...
```

### ### Combinatoric iterators

```
```python
>>> combinations('abc', 2)
[('a', 'b'), ('a', 'c'), ('b', 'c')]

>>> combinations_with_replacement('abc', 2)
[('a', 'a'), ('a', 'b'), ('a', 'c'),
 ('b', 'b'), ('b', 'c'),
 ('c', 'c')]

>>> permutations('abc', 2)
[('a', 'b'), ('a', 'c'),
 ('b', 'a'), ('b', 'c'),
 ('c', 'a'), ('c', 'b')]

>>> product('ab', [1, 2])
[('a', 1), ('a', 2),
 ('b', 1), ('b', 2)]

>>> product([0, 1], repeat=3)
[(0, 0, 0), (0, 0, 1), (0, 1, 0), (0, 1, 1),
 (1, 0, 0), (1, 0, 1), (1, 1, 0), (1, 1, 1)]
...
```

Infinite iterators


```
```python
>>> i = count(5, 2)
>>> next(i), next(i), next(i)
(5, 7, 9)

>>> a = cycle('abc')
>>> [next(a) for _ in range(10)]
['a', 'b', 'c', 'a', 'b', 'c', 'a', 'b', 'c', 'a']

>>> repeat(10, 3)
[10, 10, 10]
```

### Iterators
```python
>>> chain([1, 2], range(3, 5))
[1, 2, 3, 4]

>>> compress('abc', [True, 0, 1])
['a', 'c']

>>> # islice(<collection>, from_inclusive, to_exclusive)
>>> islice([1, 2, 3], 1, None)
[2, 3]

>>> people = [{'id': 1, 'name': 'Bob'},
 {'id': 2, 'name': 'Bob'},
 {'id': 3, 'name': 'Peter'}]
>>> groups = groupby(people, key=lambda a: a['name'])
>>> {name: list(group) for name, group in groups}
{'Bob': [{'id': 1, 'name': 'Bob'},
 {'id': 2, 'name': 'Bob'}],
 'Peter': [{'id': 3, 'name': 'Peter'}]}
```
```

Introspection and Metaprograming

****Inspecting code at runtime and code that generates code. You can:****

- * ****Look at the attributes****
- * ****Set new attributes****
- * ****Create functions dynamically****
- * ****Traverse the parent classes****
- * ****Change values in the class****

Variables

```
```python
<list> = dir() # Names of in-scope variables.
<dict> = locals() # Dict of local variables. Also vars().
<dict> = globals() # Dict of global variables.
```
```

Attributes

```
```python
class Z:
```

```
def __init__(self):
 self.a = 'abcde'
 self.b = 12345
...

```python
>>> z = Z()

>>> vars(z)
{'a': 'abcde', 'b': 12345}

>>> getattr(z, 'a')
'abcde'

>>> hasattr(z, 'c')
False

>>> setattr(z, 'c', 10)
...

### Parameters
```python
from inspect import signature
sig = signature(<function>)
no_of_params = len(sig.parameters)
param_names = list(sig.parameters.keys())
...

Type
**Type is the root class. If only passed the object it returns it's type. Otherwise it
creates a new class (and not the instance!).**

```python
type(<class_name>, <parents_tuple>, <attributes_dict>)
...

```python
>>> Z = type('Z', (), {'a': 'abcde', 'b': 12345})
>>> z = Z()
...

Meta Class
Class that creates class.

```python
def my_meta_class(name, parents, attrs):
    attrs['a'] = 'abcde'
    return type(name, parents, attrs)
...

#### Or:
```python
class MyMetaClass(type):
 def __new__(cls, name, parents, attrs):
 attrs['a'] = 'abcde'
```

```
 return type.__new__(cls, name, parents, attrs)
...

Metaclass Attribute
**When class is created it checks if it has metaclass defined. If not, it recursively
checks if any of his parents has it defined and eventually comes to type.**

```python
class MyClass(metaclass=MyMetaClass):
    def __init__(self):
        self.b = 12345
...

```

Operator

```
-----
```python
from operator import add, sub, mul, truediv, floordiv, mod, pow, neg, abs, \
 eq, ne, lt, le, gt, ge, \
 not_, and_, or_, \
 itemgetter, attrgetter, methodcaller
...

```

```
```python
import operator as op
product_of_elems = functools.reduce(op.mul, <list>)
sorted_by_second = sorted(<list>, key=op.itemgetter(1))
sorted_by_both = sorted(<list>, key=op.itemgetter(1, 0))
LogicOp = enum.Enum('LogicOp', {'AND': op.and_, 'OR' : op.or_})
last_el = op.methodcaller('pop')(<list>)
...

```

Eval

```
-----
### Basic
```python
>>> from ast import literal_eval
>>> literal_eval('1 + 2')
3
>>> literal_eval('[1, 2, 3]')
[1, 2, 3]
>>> ast.literal_eval('abs(1)')
ValueError: malformed node or string
...

```

#### ### Using Abstract Syntax Trees

```
```python
import ast
from ast import Num, BinOp, UnaryOp
import operator as op

LEGAL_OPERATORS = {ast.Add:    op.add,
                   ast.Sub:    op.sub,
                   ast.Mult:   op.mul,

```

```
        ast.Div:      op.truediv,
        ast.Pow:      op.pow,
        ast.BitXor:   op.xor,
        ast.USub:     op.neg}

def evaluate(expression):
    root = ast.parse(expression, mode='eval')
    return eval_node(root.body)

def eval_node(node):
    node_type = type(node)
    if node_type == Num:
        return node.n
    if node_type not in [BinOp, UnaryOp]:
        raise TypeError(node)
    operator_type = type(node.op)
    if operator_type not in LEGAL_OPERATORS:
        raise TypeError(f'Illegal operator {node.op}')
    operator = LEGAL_OPERATORS[operator_type]
    if node_type == BinOp:
        left, right = eval_node(node.left), eval_node(node.right)
        return operator(left, right)
    elif node_type == UnaryOp:
        operand = eval_node(node.operand)
        return operator(operand)
    ...

```python
>>> evaluate('2 ^ 6')
4
>>> evaluate('2 ** 6')
64
>>> evaluate('1 + 2 * 3 ** (4 ^ 5) / (6 + -7)')
-5.0
```
```

Coroutine

```
* **Similar to Generator, but Generator pulls data through the pipe with iteration, while
Coroutine pushes data into the pipeline with send().**
* **Coroutines provide more powerful data routing possibilities than iterators.**
* **If you built a collection of simple data processing components, you can glue them
together into complex arrangements of pipes, branches, merging, etc.**
```

Helper Decorator

```
* **All coroutines must be "primed" by first calling next().**
* **Remembering to call next() is easy to forget.**
* **Solved by wrapping coroutines with a decorator:**
```

```
```python
def coroutine(func):
 def out(*args, **kwargs):
 cr = func(*args, **kwargs)
 next(cr)
 return out
```

```
 return cr
 return out
...

Pipeline Example
```python
def reader(target):
    for i in range(10):
        target.send(i)
    target.close()

@coroutine
def adder(target):
    while True:
        item = (yield)
        target.send(item + 100)

@coroutine
def printer():
    while True:
        item = (yield)
        print(item)

reader(adder(printer())) # 100, 101, ..., 109
...

```


Libraries
=====

Progress Bar

```
-----
```python
$ pip3 install tqdm
from tqdm import tqdm
from time import sleep
for i in tqdm([1, 2, 3]):
 sleep(0.2)
for i in tqdm(range(100)):
 sleep(0.02)
...

```

Plot

```

```python
# $ pip3 install matplotlib
from matplotlib import pyplot
pyplot.plot(<data_1> [, <data_2>, ...])
pyplot.savefig(<filename>, transparent=True)
pyplot.show()
...

```

Argparse

```
-----
```python
from argparse import ArgumentParser
desc = 'calculate X to the power of Y'
parser = ArgumentParser(description=desc)
group = parser.add_mutually_exclusive_group()
group.add_argument('-v', '--verbose', action='store_true')
group.add_argument('-q', '--quiet', action='store_true')
parser.add_argument('x', type=int, help='the base')
parser.add_argument('y', type=int, help='the exponent')
args = parser.parse_args()
answer = args.x ** args.y

if args.quiet:
 print(answer)
elif args.verbose:
 print(f'{args.x} to the power {args.y} equals {answer}')
else:
 print(f'{args.x}^{args.y} == {answer}')
...

```

## Table

```

Prints CSV file as ASCII table:
```python
# $ pip3 install tabulate
import csv
from tabulate import tabulate
with open(<filename>, encoding='utf-8') as file:
    lines = csv.reader(file, delimiter=';')
    headers = [header.title() for header in next(lines)]
    table = tabulate(lines, headers)
    print(table)
...

```

Curses

```
-----
```python
$ pip3 install curses
from curses import wrapper

def main():
 wrapper(draw)

def draw(screen):
 screen.clear()
 screen.addstr(0, 0, 'Press ESC to quit.')
 while screen.getch() != 27:
 pass

def get_border(screen):
 from collections import namedtuple

```

```

P = namedtuple('P', 'x y')
height, width = screen.getmaxyx()
return P(width - 1, height - 1)
...

```

## Image

-----

```

Creates PNG image of greyscale gradient:
```python
# $ pip3 install pillow
from PIL import Image
width = 100
height = 100
size = width * height
pixels = [255 * i/size for i in range(size)]

img = Image.new('L', (width, height), 'white')
img.putdata(pixels)
img.save('test.png')
```

```

```

Modes
* **`'1'` - 1-bit pixels, black and white, stored with one pixel per byte.**
* **`'L'` - 8-bit pixels, greyscale.**
* **`'RGB'` - 3x8-bit pixels, true color.**
* **`'RGBA'` - 4x8-bit pixels, true color with transparency mask.**
* **`'HSV'` - 3x8-bit pixels, Hue, Saturation, Value color space.**

```

## Audio

-----

```

Saves a list of floats with values between -1 and 1 to a WAV file:
```python
import wave, struct
samples = [struct.pack('<h', int(a * 30000)) for a in <list>]
wf = wave.open('test.wav', 'wb')
wf.setnchannels(1)
wf.setsampwidth(2)
wf.setframerate(44100)
wf.writeframes(b''.join(samples))
wf.close()
```

Plays Popcorn
```python
# pip3 install simpleaudio
import simpleaudio, math, struct
from itertools import chain, repeat
F = 44100
S1 = '71♪,69,,71♪,66,,62♪,66,,59♪,,, '
S2 = '71♪,73,,74♪,73,,74,,71,,73♪,71,,73,,69,,71♪,69,,71,,67,,71♪,,, '
get_pause = lambda seconds: repeat(0, int(seconds * F))
sin_f = lambda i, hz: math.sin(i * 2 * math.pi * hz / F)
get_wave = lambda hz, seconds: (sin_f(i, hz) for i in range(int(seconds * F)))

```

```
get_hz    = lambda n: 8.176 * 2 ** (int(n) / 12)
parse_n   = lambda note: (get_hz(note[:2]), 0.25 if len(note) > 2 else 0.125)
get_note  = lambda note: get_wave(*parse_n(note)) if note else get_pause(0.125)
samples_f = chain.from_iterable(get_note(n) for n in f'{S1}{S1}{S2}'.split(','))
samples_b = b''.join(struct.pack('<h', int(a * 30000)) for a in samples_f)
simpleaudio.play_buffer(samples_b, 1, 2, F)
...
```

Url

```
---
```python
from urllib.parse import quote, quote_plus, unquote, unquote_plus
...
```

### Encode

```
```python
>>> quote("Can't be in URL!")
'Can%27t%20be%20in%20URL%21'
>>> quote_plus("Can't be in URL!")
'Can%27t+be+in+URL%21'
...
```

Decode

```
```python
>>> unquote('Can%27t+be+in+URL%21')
"Can't+be+in+URL!"
>>> unquote_plus('Can%27t+be+in+URL%21')
"Can't be in URL!"
...
```

Scraping

```

```python
# $ pip3 install requests beautifulsoup4
>>> import requests
>>> from bs4 import BeautifulSoup
>>> url    = 'https://en.wikipedia.org/wiki/Python_(programming_language)'
>>> page   = requests.get(url)
>>> doc    = BeautifulSoup(page.text, 'html.parser')
>>> table  = doc.find('table', class_='infobox vevent')
>>> rows   = table.find_all('tr')
>>> link   = rows[11].find('a')['href']
>>> ver    = rows[6].find('div').text.split()[0]
>>> link, ver
('https://www.python.org/', '3.7.2')
...
```

Web

```
---
```python
$ pip3 install bottle
from bottle import run, route, post, template, request, response
```



```
import json
'''

Run
```python
run(host='localhost', port=8080)
run(host='0.0.0.0', port=80, server='cherrypy')
'''

### Static Request
```python
@route('/img/<image>')
def send_image(image):
 return static_file(image, 'images/', mimetype='image/png')
'''

Dynamic Request
```python
@route('/<sport>')
def send_page(sport):
    return template('<h1>{{title}}</h1>', title=sport)
'''

### REST Request
```python
@post('/odds/<sport>')
def odds_handler(sport):
 team = request.forms.get('team')
 home_odds, away_odds = 2.44, 3.29
 response.headers['Content-Type'] = 'application/json'
 response.headers['Cache-Control'] = 'no-cache'
 return json.dumps([team, home_odds, away_odds])
'''

Test:
```python
# $ pip3 install requests
>>> import requests
>>> url = 'http://localhost:8080/odds/football'
>>> data = {'team': 'arsenal f.c.'}
>>> response = requests.post(url, data=data)
>>> response.json()
['arsenal f.c.', 2.44, 3.29]
'''
```

Profile

```
### Basic
```python
from time import time
start_time = time() # Seconds since Epoch.
...
duration = time() - start_time
'''
```

```

High Performance
```python
from time import perf_counter as pc
start_time = pc()    # Seconds since restart.
...
duration = pc() - start_time
```

Timing a Snippet
```python
from timeit import timeit
timeit('"-".join(str(a) for a in range(100))',
        number=10000, globals=globals(), setup='pass')
...

### Line Profiler
```python
$ pip3 install line_profiler
@profile
def main():
 a = [*range(10000)]
 b = {*range(10000)}
main()
```

#### Usage:
```text
$ kernprof -lv test.py
Line # Hits Time Per Hit % Time Line Contents
=====
 1 @profile
 2 def main():
 3 1 1128.0 1128.0 27.4 a = [*range(10000)]
 4 1 2994.0 2994.0 72.6 b = {*range(10000)}
...

Call Graph
Generates a PNG image of call graph with highlighted bottlenecks:

```python
# $ pip3 install pycallgraph
from pycallgraph import output, PyCallGraph
from datetime import datetime
time_str = datetime.now().strftime('%Y%m%d%H%M%S')
filename = f'profile-{time_str}.png'
drawer = output.GraphvizOutput(output_file=filename)
with PyCallGraph(output=drawer):
    <code_to_be_profiled>
...

NumPy
-----
**Array manipulation mini language. Can run up to 100 times faster than equivalent Python

```

```
code.**

```python
$ pip3 install numpy
import numpy as np
```

```python
<array> = np.array(<list>)
<array> = np.arange(from_inclusive, to_exclusive, step_size)
<array> = np.ones(<shape>)
<array> = np.random.randint(from_inclusive, to_exclusive, <shape>)
```

```python
<array>.shape = <shape>
<view> = <array>.reshape(<shape>)
<view> = np.broadcast_to(<array>, <shape>)
```

```python
<array> = <array>.sum(<axis>)
indexes = <array>.argmin(<axis>)
```

* **Shape is a tuple of dimension sizes.**
* **Axis is an index of dimension that gets collapsed.**

### Indexing
```bash
<el> = <2d_array>[0, 0] # First element.
<1d_view> = <2d_array>[0] # First row.
<1d_view> = <2d_array>[:, 0] # First column. Also [..., 0].
<3d_view> = <2d_array>[None, :, :] # Expanded by dimension of size 1.
```

```bash
<1d_array> = <2d_array>[<1d_row_indexes>, <1d_column_indexes>]
<2d_array> = <2d_array>[<2d_row_indexes>, <2d_column_indexes>]
```

```bash
<2d_bools> = <2d_array> > 0
<1d_array> = <2d_array>[<2d_bools>]
```

* **If row and column indexes differ in shape, they are combined with broadcasting.**

### Broadcasting
**Broadcasting is a set of rules by which NumPy functions operate on arrays of different sizes and/or dimensions.**

```python
left = [[0.1], [0.6], [0.8]] # Shape: (3, 1)
right = [0.1 , 0.6 , 0.8] # Shape: (3)
```

```
...
```

```
1. If array shapes differ, left-pad the smaller shape with ones:
```

```
```python
left  = [[0.1], [0.6], [0.8]] # Shape: (3, 1)
right = [[0.1 , 0.6 , 0.8]] # Shape: (1, 3) <- !
```
```

```
2. If any dimensions differ in size, expand the ones that have size 1 by duplicating
their elements:
```

```
```python
left  = [[0.1, 0.1, 0.1], [0.6, 0.6, 0.6], [0.8, 0.8, 0.8]] # Shape: (3, 3) <- !
right = [[0.1, 0.6, 0.8], [0.1, 0.6, 0.8], [0.1, 0.6, 0.8]] # Shape: (3, 3) <- !
```
```

```
3. If neither non-matching dimension has size 1, rise an error.
```

```
Example
```

```
For each point returns index of its nearest point (`[0.1, 0.6, 0.8] => [1, 2, 1]`):
```

```
```python
>>> points = np.array([0.1, 0.6, 0.8])
[ 0.1, 0.6, 0.8]
>>> wrapped_points = points.reshape(3, 1)
[[ 0.1],
 [ 0.6],
 [ 0.8]]
>>> distances = wrapped_points - points
[[ 0. , -0.5, -0.7],
 [ 0.5,  0. , -0.2],
 [ 0.7,  0.2,  0. ]]
>>> distances = np.abs(distances)
[[ 0. , 0.5, 0.7],
 [ 0.5, 0. , 0.2],
 [ 0.7, 0.2, 0. ]]
>>> i = np.arange(3)
[0, 1, 2]
>>> distances[i, i] = np.inf
[[ inf, 0.5, 0.7],
 [ 0.5, inf, 0.2],
 [ 0.7, 0.2, inf]]
>>> distances.argmin(1)
[1, 2, 1]
```
```

```
Basic Script Template
```

```

```

```
```python
#!/usr/bin/env python3
#
# Usage: .py
#
```

```
from collections import namedtuple
from enum import Enum
import re
import sys

def main():
    pass

###
##  UTIL
#

def read_file(filename):
    with open(filename, encoding='utf-8') as file:
        return file.readlines()

if __name__ == '__main__':
    main()

...
```